MA 3046 - Matrix Analysis
Laboratory Number 9
Power Methods for the Eigenvalue Problem

As we have previously discussed, the eigenvalue problem:

$$\mathbf{A}\,\mathbf{x} \;=\; \lambda\,\mathbf{x}$$

where $\mathbf{A}$ is an $m \times m$ (square) matrix, has a very rich theory associated with it.

Unfortunately, the eigenvalue problem also poses a significant real computational challenge. The traditional lower-division course approach(es) to this problem, i.e. solving the characteristic equation,

$$\mathrm{P}_m(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I}) = 0$$

become computationally infeasible for "real" matrices, due to both the cost of computing determinants and to the potential instability of the roots of high degree polynomials.

The computational solution of real eigenvalue problems, i.e. problems that cannot be conveniently done by hand, generally involves either:

(1) *Power Methods*, which can generally and relatively "inexpensively" find one or a "few" eigenvectors and the associated eigenvalues, and
(2) *Similarity Transformation-Based Methods*, which can find all of the eigenvalues and eigenvectors, but generally at fairly high "cost."

In this laboratory we shall study pertinent aspects of both classes of practical methods.

Power Methods look for what is, in some sense, the "dominant" (e.g. largest, smallest, closest to some prescribed value) eigenvalue of a given matrix. They utilize repeated multiplications to produce a sequence of iterates which actually converge to eigenvector associated with the dominant eigenvalue, from which the associated eigenvalue is easily determined. In order to ensure the successful convergence of these algorithms, two restrictions are needed:

(1) The "dominant" eigenvalue must be *unique*, and
(2) The iterate must be *normalized* after each step to prevent over/underflow.

The most-commonly encountered variants of Power Method algorithms are:

The Basic Power Method: Do until convergence:

$$\mathbf{x}^{(k+1)} = \mathbf{A}\mathbf{x}^{(k)}$$
$$\mathbf{x}^{(k+1)} = \frac{\mathbf{x}^{(k+1)}}{\|\mathbf{x}^{(k+1)}\|} \quad .$$

The Shifted Inverse Power Method: For λ^* fixed, do until convergence:

$$(\mathbf{A} - \lambda^* \mathbf{I})\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$$

$$\mathbf{x}^{(k+1)} = \frac{\mathbf{x}^{(k+1)}}{\|\mathbf{x}^{(k+1)}\|} \quad .$$

In these algorithms, convergence is assumed to mean to some (reasonable) number of preset significant digits, and, since eigenvectors are not unique, any appropriate norm can be used for the normalization step. (Note, in MATLAB, we can use any of the norms we have been frequently using in class, since $\|\mathbf{x}\|_1$, $\|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_\infty$ are implemented with the functions:

norm(x,1) , **norm(x)** and **norm(x,'inf')**

respectively. Also note that some care must be taken to identify convergence when the effective largest eigenvalue is negative, since in this case power method iterate vectors reverse direction each step. This last unpleasantness can be avoided if we divide by the (signed) value of the component of $\mathbf{x}^{(k)}$ with the largest magnitude, instead of the norm. But, of course, this avoidance comes at the cost of additional coding complexity.) An implementation of the power method for a simple matrix is contained in program **powermeth.m**, which is shown in Figure 9.1

```
%              This script implements the power method to find the
%       dominant eigenvector of an already given matrix.
%
      [ rowsA, colsA ]   =   size(a);
      x          =   ones(rowsA,1);
      data  =  [  0   x'  ] ;
%
      for  n = 1:10 ;
              x     = a*x ;
              x     = x/norm(x);
              data = [ data ; n  x' ] ;
      end
%
      data
```

Figure 9.1 - Listing of Program **powermeth.m**

Of course, once one has found an eigenvector, they still need the associated eigenvalue (and vice versa). Finding the eigenvalue associated with a given eigenvector, however, is relatively trivial. For example, one can compute $\mathbf{A}\,\mathbf{x}$, and then simply compare the ratio of the nonzero components, since, by definition, that ratio must be λ. Alternatively, and somewhat more elegantly, one may also view the eigenvalue problem as the overdetermined system of n equations in one unknown:

$$\mathbf{x}\,\lambda \;=\; \mathbf{A}\,\mathbf{x}$$

and determine λ by the standard least-squares, projection approach, i.e.

$$\lambda \;=\; \left(\mathbf{x}^H\mathbf{x}\right)^{-1}\mathbf{x}^H\mathbf{A}\,\mathbf{x} \;\equiv\; \frac{\mathbf{x}^H\mathbf{A}\,\mathbf{x}}{\mathbf{x}^H\mathbf{x}}$$

As we have seen in class, MATLAB provides basically only one eigenvalue routine, the built-in (i.e. not an **m**-file) command **eig( )**. While we shall defer to the next laboratory a detailed study of the algorithm underlying this program, we shall use the routine here as a "black box."

Unfortunately, power methods have a potentially serious drawback. This is that their effective rate of convergence is linear, and depends on the ratio between the largest (effective) eigenvalue to the second-largest. More specifically, in the case of the shifted inverse power method, the rate depends on the ratio

$$\frac{|\lambda_K - \lambda^*|}{|\lambda_J - \lambda^*|}$$

in the case of the shifted inverse power method, where λ_J is the eigenvalue closest to λ^*, and λ_K the next closest. This latter observation implies that convergence can be dramatically increased by improving the estimated eigenvalue λ^*. Moreover, in the case of a real, symmetric matrix, we can show that

$$\lambda_1 = \max_{\mathbf{x}} \;\frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}$$

These observations lead to:

Rayleight Quotient Iteration: Do until convergence:

$$\lambda^* = \frac{{\mathbf{x}^{(k)}}^T\mathbf{A}\mathbf{x}^{(k)}}{{\mathbf{x}^{(k)}}^T\mathbf{x}^{(k)}}$$

$$(\mathbf{A}-\lambda^*\mathbf{I})\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} \quad .$$

However, this method may not be as reliable for matrices which are not real and symmetric. But, on the bright side, a tremendous number of practical applications do involve only the real, symmetric case.

[This Page Intentionally Left Blank]

Name: _____

MA 3046 - Matrix Analysis
Laboratory Number 9
Power Methods for the Eigenvalue Problem

1. Link to the laboratory web page and download the files:

powermeth.m, **time_eig.m** and **eigstuff.mat**

to your laboratory directory.

2. Start MATLAB and give the command

load eigstuff

.

Examine the matrix:

apow = $\begin{bmatrix} & \\ & \\ & \\ & \\ & \\ & \end{bmatrix}$

Does this matrix appear to have an particular structure that would allow us to predict the structure of its eigenvalues or eigenvectors? If so, what?

Do the actual eigenvalues as computed by **eig()** confirm your above conclusion?

3. Open the program **powermeth.m** in a text editor window, and study it until you feel you understand its logic, and are convinced that it is properly implementing the basic power method. The run it, with the matrix **a** equal to **apow**, until convergence to four significant digits is obtained. Record the solution:

$$\mathbf{x} = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} \quad \text{number of iterations} = \underline{\qquad\qquad}$$

4. Given the converged eigenvector ($\mathbf{x}$) determined above, estimate the largest eigenvalue of the matrix $\mathbf{A}$.

Solution: $\lambda = \underline{\qquad\qquad}$

5. Use any text editor, modify program **powermeth.m** by changing the line

$$\mathbf{x = a * x ;}$$

to

$$\mathbf{x = a \backslash x ;}$$

What should be the effect of this change. (You may want to save this under a different name than **powermeth.m**.

6. Using the script created above, iterate until four significant digit convergence is obtained. (You may have to increase the loop counter to achieve this!) Record the solution:

$$\mathbf{x} = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix} \quad \text{number of iterations} = \underline{\qquad\qquad}$$

7. Given the converged eigenvector ($\mathbf{x}$) determined above, estimate the associated eigenvalue of the matrix $\mathbf{A}$.

Solution: $\lambda = \underline{\qquad\qquad}$

8. Use any text editor again, change the line in the script:

$$\mathbf{x = a \setminus x ;}$$

to

$$\mathbf{x = (a - 0.5*eye(6)) \setminus x ;}$$

What iteration should the script now perform?

9. Run this latest revision of the script until four significant digit convergence is again obtained. (You may have to adjust the number of iterations!) Record the solution:

$$\mathbf{x} = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} \quad \text{number of iterations} = \underline{\qquad\qquad}$$

10. Given the latest converged eigenvector ($\mathbf{x}$) determined above, estimate the associated eigenvalue of the matrix $\mathbf{A}$.

Solution: $\lambda = \underline{\qquad\qquad}$

11. Compare your estimated eigenvalues from parts 4, 7 and 10 with the values given by the MATLAB function **eig(a)**. Do they reasonably correspond? Is the one found in part 10 in fact the one closest to $\lambda^* = 0.5$?

12. What do you think would happen, both in theory and in finite precision arithmetic, if your script had **not** reset the vector $\mathbf{x}$ to all ones prior to running either part 7 or part 10 above?

Change the code and confirm or refute your hypothesis above.

13. Clear the MATLAB workspace. Then generate a random 100×100 *symmetric* matrix. (Hint - note $\mathbf{A} + \mathbf{A}^H$ is always symmetric.) Then using the starting vector of

$$\mathbf{x}^{(0)} = [\ 1\ \ 1\ \ 1\ \cdots\ 1\ ]^T$$

repeat Rayleigh quotient iteration until you converge to an eigenvalue. (A single iteration should require one short line of code!) Record the value and the number of iterations required.

14. Do **not** change the matrix you used in part 13, but generate a new $\mathbf{x}^{(0)}$ as a random 100×1 vector. Again repeat Rayleigh quotient iteration until you converge to an eigenvalue. Record the value and the number of iterations required.

How does this eigenvalue compare with the one determined in part 13? If they are different, why?